AFRL-IF-RS-TR-2003-170
Final Technical Report
July 2003

THE SERRANO PROJECT FINAL REPORT: NON-INVASIVELY RETROFITTING LEGACY APPLICATIONS TO WITHSTAND INTRUSIONS

University of Texas at Austin

Sponsored by
Defense Advanced Research Projects Agency
DARPA Order No. E284

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency or the U.S. Government.

AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE
ROME RESEARCH SITE
ROME, NEW YORK

This report has been reviewed by the Air Force Research Laboratory, Information Directorate, Public Affairs Office (IFOIPA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

AFRL-IF-RS-TR-2003-170 has been reviewed and is approved for publication.

APPROVED: Robert J. Vaeth

ROBERT J. VAETH
Project Engineer

FOR THE DIRECTOR:

WARREN H. DEBANY JR., Technical Advisor
Information Grid Division
Information Directorate

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 074-0188

1. AGENCY USE ONLY (Leave blank)	**2. REPORT DATE** JULY 2003	**3. REPORT TYPE AND DATES COVERED** Final Aug 96 – Nov 99

4. TITLE AND SUBTITLE
THE SERRANO PROJECT FINAL REPORT: NON-INVASIVELY RETROFITTING LEGACY APPLICATIONS TO WITHSTAND INTRUSIONS

5. FUNDING NUMBERS
C - F30602-96-1-0313
PE - 62301E
PR - E017
TA - 01
WU - 03

6. AUTHOR(S)
Aleta Ricciardi and Keith Marzullo

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
University of Texas at Austin
Office of Sponsored Projects
PO Box 7726
Austin Texas 78712-7726

8. PERFORMING ORGANIZATION REPORT NUMBER
N/A

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Defense Advanced Research Projects Agency
3701 North Fairfax Drive
Arlington Virginia 22203-1714

AFRL/IFGB
525 Brooks Road
Rome New York 13441-4505

10. SPONSORING / MONITORING AGENCY REPORT NUMBER
AFRL-IF-RS-TR-2003-170

11. SUPPLEMENTARY NOTES

AFRL Project Engineer: Robert J. Vaeth/IFGB/(315) 330-2182/ Robert.Vaeth@rl.af.mil

12a. DISTRIBUTION / AVAILABILITY STATEMENT
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

12b. DISTRIBUTION CODE

13. ABSTRACT ***(Maximum 200 Words)***

The Serrano project's goals were to develop new techniques for making secure and reliable distributed systems. There was a special focus on legacy environments and exploiting modern, object oriented techniques to incorporate them in distributed applications based on secure, reliable middleware. The final results arose in five general areas: the implication of reliable multicast on further transmitting system infection, making safe progress in a distributed application despite the presence of network partitions, mechanism for making systems intrusion tolerant, efficient epidemiological update protocols, and software engineering approaches for automating the design of fault-tolerant distributed applications.

14. SUBJECT TERMS
Security, Reliable Middleware, Distributed Applications, Distributed Systems, Legacy Environments, Object Oriented Techniques, Reliable Multicast, Intrusion Tolerant, Protocols, Software Engineeering

15. NUMBER OF PAGES
15

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	**18. SECURITY CLASSIFICATION OF THIS PAGE**	**19. SECURITY CLASSIFICATION OF ABSTRACT**	**20. LIMITATION OF ABSTRACT**
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

Table of Contents

1. RESEARCH RESULTS

Our results arose in five general areas: the implication of reliable multicast on further transmitting system infection, making safe progress in a distributed application despite the presence of network partitions, mechanism for making systems intrusion tolerant, efficient epidemiological update protocols, and software engineering approaches for automating the design of fault-tolerant distributed applications. We summarize our results below and refer interested parties to our publications (listed in Section 3).

1.1 Infection and Reliable Multicast

There are similarities between problems associated with processes that are under the control of an intruder and problems associated with processes that are arbitrarily faulty. A process that is under the control of an intruder may masquerade as a legitimate process and, like an arbitrarily faulty process, may not follow the specification that other processes expect it to.

Given this similarity, it seems plausible to mask the effects of such compromised processes in the same way that one masks arbitrary failures. Masking the effects of failures requires replication, and several protocols have in fact been designed to use replication to mask the effects of such processes, such as the ISIS, Ensemble, Totem and Rampart projects. The bounds for masking arbitrary failures hold for these protocols, such as the need for either digital signatures or $(3f+1)$-fold replication in order to mask f compromised processes when reaching agreement.

However, an intruder may wreak more damage than what is captured by the arbitrary failure model. For example, an intruder may launch a malicious attack towards other processes on the system. It can create other seemingly benign processes by exploiting *transitive trust* that is assumed with the use of, for example, a *.rhosts* file, or it can co-opt otherwise correct processes through mechanisms like trap doors and race condition attacks. This implies that the techniques used to mask arbitrarily faulty processes may not be applicable, because too many processes may become compromised thereby violating the replication assumption. Accepting that the natural occurrence of Byzantine faults is small, and that one likely source for such faults is due to malicious attacks, then the self-propagating nature of these attacks should also be considered.

Hence, we studied how different multicast strategies effect the efficacy of such attacks. We modeled these attacks as a simple form of infection. We assumed that intruders can infect processes with a given probability by sending it a message. We considered only the messages in multicast strategies that carry the user's data, since these are the messages over which an application process has the most control.

We measure this effect in terms of *availability*, which is the probability that no more than a certain number processes are infected. We consider the two questions "what is the availability of the system after having run for some period of time?" We examine how the answers to these questions change as the number of processes grows, as the probability of a message being infective changes, and as different multicast strategies are used.

The results of our work impact the design of middleware for group-based communication systems. An attack at this level of a system would be very hard to detect. Using our results, one could use a standard protocol for masking arbitrary failures. Given a desired availability, one can determine how long this protocol can run before the processes must be "cleaned", either by restarting processes from known clean images or by running a diagnostic program. This periodic cleaning ensures that the initial assumption of no more than f processes being arbitrarily faulty is maintained with an acceptable level of likelihood.

Our results are applicable to more than group-based communications. For example, in a mobile agent system, one can model a compromised landing pad as an infected process, and a mobile agent capable of compromising a land pad as a message from an infected process. In this context, availability is the probability that no more than a given number of landing pads are compromised. if one uses mobile agents to collect information in a web-crawl-like manner, a compromised landing pad can corrupt the information that agents carry while passing through that pad. The results of this paper can be used to choose a dissemination mechanism for the agents and decide how often the landing pads must be "cleaned".

From a modeling point of view, our work resembles epidemiological modeling of algorithms. Our model is essentially that of a simple epidemic with a zero latency period. If cleaning were also modeled, then one would have a general epidemic. There has been work on modeling the spread of computer viruses as a general epidemic. However, our work differs from existing epidemiological approaches in several aspects. First, we are interested in how the infection process affects the availability of the system rather than the expected number of infected processes. This is important because we wish to apply the results to multicast protocols. If one builds a protocol that can mask f arbitrarily faulty processes, then one would like to know how likely it is that this assumption holds. Knowing the average number of infected processes does not address this question. Second, our transmission of infection is more restricted than general mixing of populations or a mixing restricted to undirected graphs. Third, we separate infection from death because death (for us, cleaning by restarting processes from trusted object files) is not a stochastic process. Instead, cleaning is periodically initiated based on the rate of infection and the desired availability, which ensures that any initial assumptions regarding the maximum number of (arbitrarily) faulty processes is maintained with an acceptable likelihood. At a pragmatic level, cleaning is expensive, and so should only be done as infrequently as possible rather than at random times chosen from some distribution.

1.2 Withstanding Partitions

Intrusions and viruses can cause large segments of a distributed systems to be either completely unavailable or of questionable trustworthiness. In this way, intrusions and viruses can cause a distributed system to *partition*, just as network failures and quality of service outages do. However, partitions caused by intrusions and viruses need not be geographically localized, but rather affect whole classes of platforms, operating system, and/or applications. We examined how distributed applications can make safe progress despite the presence of partitions across wide geographies. This includes group management and membership in partitionable systems, upon which strategies for safe application progress are based.

There exist several specifications, protocols and implementations for group membership in systems that can suffer partitions. Informally, there is a set of core properties that they all share, but they differ in the exact properties that they provide. These systems are meant to provide a basis for implementation of what has been called *partition-aware* applications, which are applications that are able to make progress in multiple concurrent partitions (that is, in multiple *connected components*) without blocking.

An essential problem confronted when building any distributed system is the uncertainty at any process of the global state. Partition-aware applications are especially sensitive to this problem because actions taken in one connected component cannot be detected by the processes outside of the component. Furthermore, when communication failures cause the system to partition, the processes may not agree at the point in the history that the partition occurred. The first issues must be directly addressed by the application, and partitionable group membership protocols help processes address the second issue.

We examined a particular partition-aware application to evaluate the properties provided by different partitionable group membership protocols. The application we examine is a simple resource allocation problem that we call the *Bancomat* problem. We defined a metric specific to this application, which we call the *cushion*, which captures the effects of the uncertainty of the global state cased from partitioning. The cushion is not the only interesting metric for the *Bancomat* problem. Other metrics, such as message complexity, message size, and latency are important. However, the cushion metric does reflect an effect of the uncertainty of message delivery. The solutions we developed used different properties of partitionable group membership protocols. Thus, indirectly, the cushion of a solution also gives a measure of how well a given partitionable group membership protocol addresses uncertainty in the global state.

We were not the first to specify and examine this useful partition-aware application. We are not the first to consider this application but none before us were precise enough to allow for a comparison of the properties of partitionable group membership protocols. We also needed to compile a careful comparison of partitionable group membership protocols. We believe that our more operational approach complemented the more taxonomic comparisons which preceeded us.

We developed four different approaches to writing a partition-aware application. These approaches differ in the amount of coordination among the bancomats with respect to withdrawals and deposits. The four approaches were:

1. one in which the actions are serialized among the processes to provide tight coordination among them;
2. one in which no state is explicitly shared among the processes in the system and the processes take unilateral actions based on their local states;
3. one in which all of the processes in a connected component share the same state and the actions are tightly coordinated in the component;
4. one in which processes in a connected component share state and a process informs the other processes when it has taken an action.

Our results were quite surprising, and raised some serious criticism of the approach.

- The original definition of partition-aware is weak enough to encompass a large set of problems. We were surprised, however, at how hard it was to find a partition-aware problem that was interesting in terms of being sensitive to different partitionable group membership protocols. For example, the original paper lists four different partition-aware problems, one of which is a version of the *Bancomat* problem. We have tried for formalize the other three, but so far have had only limited success in defining a appropriate metric, like the cushion, that captures the impact of uncertainty in the global state with respect to partitionable group membership protocols. Indeed at least one of these problems requires *no* communication, and therefore does not require any of the communication properties provided by group membership.

  One open question is what other partition-aware problems exist that require or benefit from the different properties provided by the group membership protocols. If there are a large number of such problems, then there may be interesting classes of problems defined by what they require from the partitionable group membership protocols. In this case, it would be worthwhile to identify such classes to aid the choice of which partitionable group membership protocol is best for a problem. If, to the contrary, there are only a few such problems, then it might be worthwhile to design partitionable group membership protocols with these specific problems in mind.

- The four different approaches we considered illustrated some tradeoffs that we did not expect.

  The first approach we considered was not a concurrent solution. This approach shows how the relative uncertainty in the system can be constrained, but the inherent cost in such a solution. This solution suffered from a performance bottleneck due to the serialization of the authorization messages

The second approach was appealing because it used very little from the group membership service. We were surprised that one property about message delivery in groups was sufficient to lower the cushion from $\left(n\left(n-1\right)/2\right)q \ to \ \lfloor n/2 \rfloor \lceil n/2 \rceil q$. The required property does not appear to be very expensive to provide.

The state-machine-like approach also does not require much from the group membership service, but what it does require is not cheap: total message delivery order within a connected component. A totally-ordered multicast is required before every withdrawal, which implies that the latency for this protocol could be high.

The intermediate approach strikes a balance between these two, but be don't yet know the value of such a balance. Our suspicion is that is should perform better, but we have not yet tested this hypothesis.

- The differences between the different group membership protocols were most important for the intermediate approach. Using relatively weak partitionable group membership protocols resulted in a large cushion, while the other protocols allow for an optimal cushion. On the other hand, the protocol for the weak membership services was extremely conservative. The observation led us to consider other weak approaches that were not conservative (studied under a separate DARPA grant).

  It has been suggested that there are a class of applications that require the strong property that a message is delivered in the group in which it was sent. This class of application has been named *group aware*. The *Bancomat* problem is not group aware by definition, but we suspect that it shares some of the properties: without rather strong message delivery properties, one can not solve the problem with an optimal cushion. Hence, we suspect that there is a deeper principle that group awareness and the *Bancomat* problem share.

  Our experience with this problem led us to reconsider how partitionable group membership services should be presented. Many of the differences appear to be irrelevant with respect to implementing at least this partition-aware problem. Instead of concentrating on providing different properties, is worthwhile to provide more information to the application concerning the state of the system when communication fails. The fundamental problem we had to confront when designing these protocols was bounding the possible states of the processes in different connected components. Having more information might allow one to further restrict the possible states.

1.3 Intrusion Containment

Authentication systems validate the identity of subjects. Most authentication systems prevent a malicious subject masquerading as a trusted subject by relying on some secret shared by the subject and the authentication system. If the secret is leaked, or if the authentication system is bypassed, then the authenticated subject may be malicious.

Real-time intrusion detection can be used as second line of defense behind the authentication system. An intrusion detection system monitors system information for suspicious activity. Such activity might be a set of commands known to be contained in a script used to attack a system, a sequence of failed attempts to start a telnet connection on a set of machines (both examples of *misuse* detection), an unusual set of shell commands or a sequence of unexpected systems calls (both examples of *anomalous behavior* detection). The intrusion detection system might generate a general report indicating that suspicious activity has been noticed or it might identify a specific subject as behaving in a suspicious manner.

We modeled subject-specific real-time intrusion detection as a kind of failure detector, where the failure is analogous to an *arbitrary* failure. Failure detectors are useful because they allow one to use classic methods of fault-tolerance in which the system is brought to a safe state when a failure is detected. Three difficulties, however, arise with this notion of an intrusion-based failure detector.

The first difficulty is that there is a latency in detecting that a subject is an intruder. The same latency exists for failure detectors that report when a subject is arbitrarily faulty: as long as the subject does not take a wrong step with respect to its specification, it cannot be detected as being faulty. In the meantime, the faulty subject may execute several sensitive actions that are entrusted to the non-faulty subject only because its specification asserts it will not misuse this trust. Hence, required is a mechanism for containing the damage that a faulty subject does before it is detected. The mechanism that we developed for this purpose was based on *optimistic protocols*, and was packaged as a set of OrbixWeb ORB-based CORBA services.

The second difficulty is that the intrusion detection systems does not have a precise specification of the subject, and so cannot state with absolute confidence that a subject is indeed an intruder or not. Intrusion detection systems typically use various statistical measures on the monitored system information to determine some level of confidence in the subject not being an intruder. If this level becomes too low, then the report is generated. False positives (valid subjects detected as intruders) and false negatives (intruded objects not detected as intruders) will occur. The relative damage due to a false negative versus a false positive depends on the application and what is currently doing. Hence, instead of having the intrusion detection system choose a priori the confidence level that triggers an intrusion report, in our system the application can choose the required confidence level, based on its current state and upon the subject in question.

The third difficulty is that the intrusion detection system, like a failure detector, simply monitors the execution of a subject, checking for deviations from the subject's specification. Since the application sets the confidence level for a subject, the intrusion detection system could act more proactively: it could either increase its surveillance or re-authenticate the subject in question should its current confidence be too far below the required confidence level. In our system, the interface to the intrusion detection system allows for such proactive detection.

Our system, COPE, provided a set of CORBA security policies and CORBA-based services that provide intrusion tolerance in the manner discussed above. [1] The salient features of the architecture were:

- Application CORBA objects were encapsulated by a CORBA application access policy. The application access policy operation can indeed be invoked based both on permissions an on the level of confidence in the subject's identity.
- The application objects were members of the class *optimist*. An optimist can make *assumptions* that are later asserted or refuted. An application access policy makes assumptions concerning the confidence in subjects' identities on behalf of the object it wraps.
- Assumptions were CORBA objects are created by *assumption factories*. A real-time intrusion detection system provides an assumption factory. When an assumption is created, the intrusion detection system attempts to assert or refute the requested level of confidence in the subject.

1.4 Efficient Epidemiological Update

The fourth direction of inquiry was an outgrowth of the virus propagation work described in Section 1.1. However, we applied the study of efficient propagation to recovery schemes rather than pathologic behavior. In particular, we attempted to examine how to efficiently propagate "good" information such as might be used to repair state after partitions, propagate routing tables, and so forth. Traditional reliable multicasts are a heavy-handed approach to guaranteed information dissemination; we examined epidemiological, or *gossip*, protocols. In contrast to reliable multicast, gossip protocols approach delivery guarantees asymptotically, asynchronously, and may include repeat deliveries of the same message to some endpoints. However, the management overhead is negligible. Our work was aimed as devising protocols to approach reliability quickly and to minimize redundant work. We exploited network topological information to achieve both.

A generic gossip protocol running at process *p* has a structure something like following:

[1] At the time the research was performed (1997-98), the CORBA security specification was in its earliest stages. Consequently, none of the available ORBs implemented the security specification correctly or fully.

```
        when ($p$ receives a new message $m$)
        while ($p$ believes that not enough of its neighbors have received $m$) {
           $q$ = a neighbor process of $p$;
           send $m$ to $q$;
}
```

If network information is not taken into account, then each process is a neighbor of every other and it is equally likely that *p* will select a "distant" neighbor as a "near" one. Earlier work addressed this problem by having each process aware of which local area network each of its neighbors is in. A process then only rarely decides to send a gossip message to a process in another local area network. This approach is attractive because it attenuates the traffic across a router without adding any additional changes to the gossip protocol. Its drawback is that it doesn't differentiate between wide area traffic and local area traffic. The performance characteristics and the link failure probabilities are different for wide area networks and local area networks. Hence, we adopted a two-level gossip hierarchy: once level for gossip within a local area network and another level for gossip among local area networks (that is, within a wide area network).

Over the wide area, reliable gossip is based on a simple heuristic: flood messages (*i.e.*, high success achieved wit high overhead) over links that are critical, and gossip over the other links. We call the fundamental approach *direction gossip* because it is sensitive to weak or poorly connected nodes and selectives chooses more reliable communication in these directions. We created and inserted a light-weight gossip server in each LAN that directs gossip traffic to subset of its *gossip server* neighbors; that is, other gossip serves that are one routing hop away.

To measure efficacy of directional gossip, we build a simple discrete event simulator to measure the performance of directional gossip. The simulator takes as input a graph with nodes representing gossip servers and links representing internet work routers. Messages are reliably sent between gossip servers and are delivered with a time chosen from a uniform distribution. We did not, in this endeavor, model link failures or gossip server failures, and hence did not implement the aging of links.

We simulated three protocols: pure flooding, gossip with a fanout *B*, and directional gossip with a fanout *B* and a edge weight *K*, which quantifies how critical a link is deemed to be. We compared the message overheads of these three different protocols, and when interesting compared their reliability. We also measured the ability of directional gossip to accurately measure weights.

We examined the reliability of directional gossip for different values of *B* and *K*. We ran simulations on the following topologies:

1. A network of 66 nodes. This network consists of two transit domains each having on average three transit nodes. Each transit node connects to, on average, two stub domains. Each stub domain contains an average of five stub nodes. The average node degree within domains is two and is one between domains.

2. A network of 102 nodes. This has the same properties as the previous topology except that stub domains have, on average, eight nodes and each such node has, on average, a degree of five.

Since the flooding protocol always delivers messages to all nodes, it has a reliability of 1.0 when used over the critical links. Consequently, directional gossip in the 66-node WAN with *B=2* and *K=4* has a reliability of 0.9492, and in the 102-node WAN with *B=4* and *K=4* has reliability of 0.8994. In contrast, traditional (non-directional) gossip with *B=2* in the 66-node WAN has a reliability of 0 and with *B=4* in the 102-node WAN has reliability of 0.0597.

Our simulations showed that in this particular 66-node WAN, reliability was increased much more by increasing *K* than by increasing *B*. This makes sense given the generally low degree (two) of each node; *B* can be either 1 or 2. For the 102-node WAN, as well as for a WAN of two cliques, reliability is increased more by increasing *B* rather than increasing *K*.

Given the two topologies examined, overhead comparisons were predictable. Given that the 66-node WAN had such low node degree, each link was critical and the overhead was close to that of pure flooding. However, the overhead benefits of directional gossip were evident in the 102-node WAN because its average node degree was so much higher.

1.5 Automating the Design of Fault-tolerant Distribute Applications

This thread of research was borne of the desire to use existing tools for reliable computing (such as Isis and Ensemble) more efficiently. While a number of experimental toolkits exist(ed) to provide basic programming primitives for fault-tolerant distributed applications, experience shows that users often do not understand subtle differences between the various primitives. To ensure applications are correct, programmers would use the computationally costly “atomic” mechanisms that are easier to understand when lighter-weight mechanisms would have sufficed. Alternately, when programmers use primitives that are not strong enough, the program is not correct. At the time of our research, two CORBA ORBs supported replicated objects. For efficiency, they permitted programmers to specify which broadcast primitive should be used to transmit remote method invocations to the replicas of an object. These difficulties are exacerbated in the trend to deconstruct strong primitives to weaker micro-protocols. For wide-area applications, these toolkits may provide reliable point-to-point-communication, but not the robust primitives needed to build wide-area groups and ensure replica consistency. More significant though, programmers must still understand fundamental issues in fault tolerance, state transfer, asynchronous computing, replica consistency, and even thread scheduling to ensure correctness. That is, despite toolkits and distributed programming standards, programmers were still not relieved of most of the difficult issues in distributed computing.

The *Sage* project was a software development environment that assisted distributed applications programmers write correct, efficient code. Sage applied the results of a wide body of theoretical research in asynchronous distributed computing to the very real problems faced by programmers writing distributed applications. Users describe a distributed coordination problem in simple language through a series of pull-down menus. The underlying theory used the modal logics of knowledge and time, and results that detail how remote processes "learn" facts about each other's state to derive the minimal communication graph for the problem described. Sage implemented the underlying theory, provided a user-interface that shields programmers from the theory, and used the theory to derive protocols. Thus, applications programmers can use the theory to derive protocols. Thus, applications programmers can use the theory to simply and naturally, without having to learn and be proficient in it.

To further assist users understand asynchrony and fault tolerance issues and solutions, Sage allowed users to experiment with the graphical protocol by changing the order of events in the derived graph, crashing and recovering processes in the middle of a computation, dropping messages, and partitioning and repairing the network. While the initial derived protocol was shown assuming a failure-free run, users can then nonetheless examine it under the most common failure scenarios. Significantly, this experimental facility decoupled the difficulty in testing distributed applications from the effects non-deterministic, ambient system conditions have on the testing procedure itself.

2. Students

The following graduate students received support from and/or contributed to the Serrano project:

- Meng-Jang Lin, PhD 1999.
- Jeremy Sussman, PhD 1999.
- Leslie Franklin, MS 1999.
- Ramanathan Krishnamurthy, MS 1999.
- Stefano Masini, Laurea (University of Bologna), 1999 (thesis idea developed on our project).
- Paul Grisham, MS 2001.
- Chanathip Namprempre, PhD 2002 (under another project in security).

3. Publications

- A. Ricciardi. The Sage Project: A New Approach to Software Engineering for Distributed Applications. In *Proceedings of the 17th IEEE International Conference on Distributed Computing Systems*, pp244-252. May, 1997.
- M.-J. Lin, K. Marzullo, and A. Ricciardi. A New Model for Availability in the Face of Self-Propagating Attacks. In *Proceedings of the 1998 New Security Paradigms Workshop*. September, 1998.
- A. Ricciardi and P. Grisham. Toward Software Synthesis for Distributed Applications. In *Proceedings of the Seventh International Conference on Rationality and Knowledge*, pp15-28. July 1998.
- K. Marzullo and J. Sussman. The Bancomat Problem: An Example of Resource Allocation in Partitionable Asynchronous System. *Journal of Theoretical Computer Science*.
    - An earlier version of this paper appeared in 12th *International Symposium on Distributed Computing*, pp363-377. September, 1998.
- J. Y. Halpern and A. Ricciardi. A Knowledge-Theoretic Analysis of Uniform Distributed Coordination and Failure Detectors. In *Proceedings of the 18th ACM Symposium on the Principles of Distributed Computing (PODC99)*, pp73-82. May, 1999.
- C. Namprempr, J. Sussman, and K. Marzullo. Implementing causal logging using Orbix Web interception. In *Proceedings of the fifth USENIX Conference on Object-Oriented Technologies and Systems (COOTS'99)*, pp57-67. May,1999.
- M.-J. Lin and K. Marzullo. Directional gossip: gossip in a wide area network. In Dependable Computing – EDDC-3. *Third European Dependable Computing Conference*, pp364-379. September, 1999.
- L.C. Lung, J. daSilva Fraga, J-M. Farines, M. Ogg, A. Ricciardi. Cos-NamingFt – A Fault-Tolerant CORBA Naming Service. In *Proceedings of the 18th IEEE Symposium on Reliable Distributed Systems (SRDS99)*, pp254-262. September, 1999.
- M.-J. Lin, S. Masini, and K. Marzullo. Gossip versus deterministically constrained flooding on small networks. In *Fourteenth International Conference on Distributed Computing (DISC 2000)*, pp253-267. October, 2000.